101872/SYC/NAD

METHOD OF IMPLEMENTING A TREE OF DISTRIBUTED OBJECTS

The present invention concerns a method of organizing distributed objects in a hierarchy.

The invention applies to many applications employing a distributed object environment, for example telecommunication or transportation supervision applications, applications constituting an intelligent network, etc.

In a distributed object environment, an application can use different servers to provide services to clients.

The term "process" refers to a program which runs in a given environment. An object of the process is a software entity in the process.

In an application, the distributed objects are in practice organized in accordance with a given tree.

In the tree, each object has a logical name, i.e. a character string, which specifies the logical access path to that object from the starting object, i.e. the root, of the tree. The logical name is absolute in the sense that it is determined relative to the root.

It is also possible to specify the logical access path from an object other than the root. The term "relative logical name" is then used.

In any distributed object system based on an ORB, it is often necessary to access objects directly. The absolute or relative logical names of the objects must be used for this, enabling a logical access path to the required object to be found.

Furthermore, it is generally possible to ask a father object directly for access to a son object. In the request to the father object the relative logical name relative to the father object is used to designate the son object.

Because the objects are distributed, the tree of the objects in practice has many branches and the branches or parts of the branches can correspond to distinct processes. Figure 1 is a diagram showing a simplified example of an object tree.

Under a main process P0, which constitutes the root of the tree of the system, there are three distinct processes P1, P2, P3. The process P1 situated directly under the root comprises three objects, namely a first object A, which is the entry object or root of the process, from which depart two branches to two objects B and C. A final branch departs from the object B towards an object D of the process P2.

The process P3 situated directly under the root comprises a single object X.

In a practical implementation of the tree of the system the objects contain information on their respective sons. If the son objects are contained in the same process as the father object, the information consists of pointers giving the physical addresses of the son objects. If the son objects are not contained in the same process as the father object, the information is made up of references. For example, the object A contains a pointer to the object B and a pointer to the object D. The object B contains a reference to the object D.

The logical access path for accessing the object D from the root, i.e. the absolute logical name, can be written /A/B/D. It is therefore necessary to pass through the object B to arrive at the object D. This is the case whether the object D is reached by interrogating the object B directly, that object being identified by the absolute logical name /A/B, for the son object identified in the object B by a reference, or by interrogating the root for the object identified by the (absolute) logical name /A/B/D.

The objects A and B are in the process P1 and the object D is in another process P2. If the process P1 is not running, or has failed, it is no longer possible to access the object D with this tree.

Furthermore, this tree makes managing process redundancy difficult. In the example shown in figure 1, there is a redundant back-up process P2' which is substituted for the process P2 if it fails. With the tree previously explained, it is up to each object that receives a request regarding an object that is not in its process to determine to which of the two processes P2 and P2' it should transmit the request. Clearly this makes managing redundancy particularly complex. Redundant processes are routinely used to strengthen the weak points of a system, i.e. the processes that are likely to fail often, whether their failure paralyses the entire system or merely reduces the quality of service.

For the above reasons a different tree of the objects of a distributed object system has been proposed, to enable access to all the objects of the system even if some father objects are not available (failed or stopped) and to simplify the management of process redundancy. This tree employs centralized management of the tree at the level of the root, by means of a central directory, which contains structured names of all the objects. In other words, it contains all of the tree of the system.

In this tree, if a father object is interrogated for a son object, the call is redirected to the central directory. It is then always possible to access an object even

if, within the tree, that object is a son of other processes which are stopped. What is more, this also centralizes management of redundancy, which is managed by means of the same central directory.

However, this tree is very costly in terms of resources: if the number of objects is large, the central directory can be overloaded and the performance of the system seriously degraded, because of the time needed to consult the tree in the central directory for each call.

Furthermore, this solution no longer takes into account the specific nature of the distributed environment, since it treats each object identically. All calls are processed by the central directory, even if the call concerns a son object of a father object in the same process. This increases the volume of inter-process communication unnecessarily.

Finally, if the system uses the object-object protocol based on the creation of pairs of representative elements, for example the proxy/stub pairs of distributed environments based on the DCOM ORB, as the protocol for communication between objects, the centralized directory solution multiplies the number of these pairs, because it implies the creation of a pair of representative elements for each object in the tree. The pairs of representative elements are very costly in terms of memory resources.

Accordingly, an object of the invention is to provide a method of implementing a tree of distributed objects that does not have the aforementioned disadvantages.

The invention uses a central directory which contains tree information on only certain targeted objects, and so all the objects of a process can be accessed.

According to the invention, if a father object receives a location request in respect of a son object, it accesses the son object, if the latter is in the same process, or returns the call to the central directory, if it is not in the same process.

In other words, the tree within a given process is managed internally in that process, the objects of the process containing the necessary pointers to the son objects contained in the process, i.e. the physical addresses of those objects in the process concerned, but the tree of processes is managed by the central directory. This has the advantage of providing access to objects of son processes even if a father process is stopped; this enables problems of redundancy at the level of the central directory to be managed; finally, it enables the response time of the central directory to be optimized, because it has only a partial tree to manage, and it

enables the memory resources necessary for implementing the tree to be optimized.

As characterized, the invention therefore concerns a method which includes a step consisting of assigning to a father object in a process, for each son object:

- information corresponding to a physical address pB if the son object is contained in same process, or
- information referring back to said central directory if the son object is not contained in the same process.

Other features and advantages of the invention are described in the following description, which is given by way of non-limiting illustrative example only, and with reference to the accompanying drawings, in which:

- figure 1, already described, is a simplified block diagram of a prior art tree of distributed objects; and

- figure 2 is a block diagram of a tree of distributed objects in accordance with the invention.

The invention provides a central directory corresponding to the process Pr0 in figure 2. That process is the root of the tree.

There are different processes under the root process Pr0.

A first process Pr1 contains three objects A, B and C. In this process, the object A is the root object. The root object of a process is an entry object of the process. Note that there can be more than one in the same process.

The objects B and C are two son objects of the object A.

A redundant process Pr1' is a replica of the first process. In particular, it contains the same objects with the same tree.

A second process Pr2 contains two objects D and F. In this process, the object D is the root and the object F is a son object of the object D. The object D is also a son object of the object B of the process Pr1.

A redundant process Pr2' is a replica of the second process. In particular, it contains the same objects with the same tree.

The central directory contains a data structure Tab0 in which it stores information relating to the tree of the system.

In practice, it contains at least all the information relating to the entry objects (root objects) of each distinct process of the tree.

In this example, in entry E1 of the data structure there is information relating to the object A of the process Pr1: logical name relative to the central directory /A, pointer pPr1 to the corresponding process Pr1, and other information required for its

management.

In entry E2 there is information concerning the object A of the redundant process Pr1'; in entry E3 there is information on the object D of the process Pr2; in entry E4 there is information on the object D of the process Pr2'.

The central directory therefore contains the tree of the processes in the system.

According to the invention, a father object in a process (other than the central directory) contains information on its son objects, which takes the form of pointers, i.e. their physical addresses, if they are contained in the same process. Thus the object A contains pointers pB and pC to the son objects B and C, respectively.

If the son object is not in the same process, the father object contains information for returning the call to the central directory. Thus if the object B receives a request for the son object D identified by its logical name /D relative to the object B, the object B returns the request to the central directory.

In practice, it returns the request by placing the character string of its own absolute logical name, relative to the central directory, in front of the character string of the relative logical name of the object D. In the example, the absolute logical name of the object B is the character string /A/B. Accordingly, the object B transmits the request to the central directory by supplying it with the absolute logical name N(D)=/A/B/D of the object D.

If the central directory receives a request relating to an object identified by its logical name relative to the central directory, it consults its internal data structure, which is of the dictionary type, and looks up the corresponding character string. If it finds it, it obtains a corresponding reference of the object in the system. That reference enables it to transmit the request directly to the object. If it does not find it, it looks for the longest character string corresponding to a first part of the character string, in order to transmit the request to a father object for the given object identified by its relative name relative to that father object. That relative name is obtained as the reference between the two character strings. Take the example of a request received by the directory for the object C defined by its logical name N(C)=/A/C.

The central directory searches its data structure for that string or the longest string corresponding to its first part (i.e. the start of the string). In this example it will find the string /A, which is the logical name of the object A.

It therefore transmits the request relating to the object C to the object A, sending it by way of identification the relative logical name of the object C relative to

the object A. That relative logical name is simply obtained as the difference between the two character strings: /A/C - /A = /C.

In accordance with the invention, if the object to which the directory has sent a request relating to a son object cannot find the son object in its process, it sends a message to the central directory, which looks for another object in its directory. It can also place corresponding information in its data structure.

With regard to managing redundancy, figure 2 shows that the data structure Tab0 contains all the objects with the same logical names corresponding to different processes. To each entry in the table there corresponds a physical identification of the corresponding process. Thus in entry E1 there is the name /A for a process identified by a parameter pPr1 corresponding to the process Pr1. In the entry E2 there is the name /A for a process identified by a parameter pPr1' corresponding to the redundant process Pr1'.

As in the invention, as soon as an object of a process has a request relating to a son object of another process to manage, it sends its request to the central directory, which is entirely responsible for managing redundancy. In other words, it is the central directory which determines at a given time whether to send the call to the process Pr1 or to its process Pr1', depending on information on the status of the system. Thus management of redundancy is centralized.

As already mentioned, the central directory preferably contains information relating to the entry objects (root objects) of each process of the system. It therefore contains the tree of the processes (including the redundancy), while the tree in the processes is implemented internally in each process.

Finally, note that the central directory is a sensitive point of the system. Thus in practice protection mechanisms or a redundant central directory are provided in order to obtain a robust mechanism.

It has been shown that the invention applies in a distributed object environment.

One particular application concerns an environment based on a distributed Object Request Broker (ORB). ORBs known in the art include the CORBA (Common Object Request Broker Architecture) and DCOM (Distributed Component Object Mode) ORBs.